6P−8

Performance of LOTOS Execution System

Shingo Nomura†, Takashi Takizuka†, Toru Hasegawa† and Ron Greve‡
†KDD R & D Laboratories ‡University of Twente

1. Introduction

LOTOS[1] is a Formal Description Technique standardized by ISO. It is useful for specifying distributed systems, such as OSI. We have developed a LOTOS Execution System[2] which can derive an implementation from a specification in LOTOS. The derived implementation has to achieve high performance, because our purpose is to use the implementation as an actual system. The performance of the implementation strongly depends on the speed of the mechanism handling the multi-way synchronization among LOTOS processes.

This paper discusses the performance of our system, thereby focussing on the performance of the synchronization. It presents performance results obtained by using small specifications, each containing different patterns of synchronization. At the Technical University of Madrid, similar performance measurements have been carried out for another LOTOS execution system, named the TOPO compiler[3]. In this paper our results are also compared with the performance results[4] of the TOPO compiler.

2. LOTOS Execution System

The LOTOS Execution System consists of a translator and a scheduler library. The translator converts LOTOS specifications into C programs. Translated LOTOS processes are subsequently executed under control of the scheduler. A brief description of the synchronization mechanism of our system is given below.

A LOTOS process offers an event for synchronizing with other processes. The event occurs when a set of offers satisfy the synchronization conditions. In order to manage synchronization, a tree structure St (Synchronization tree) is introduced which holds the dynamic structure of synchronization. A St is created for each gate. Being offered to a gate, an event is linked to the St of that gate. Each time an event is offered, the scheduler traverses the St to detect a synchronization possibility for this event. After the first synchronization possibility is detected the event will occur. This deterministic way of detection is one of the devices[2] for achieving efficient synchronization.

Unification of data values and variables, like in PROLOG, is necessary for an event to occur. In our system, data values and variables are represented by a structure named descriptor. When the scheduler unifies variables, a new descriptor is created and the descriptors of the variables are linked to this new descriptor.

3. Performance measurements

We have measured the performance of our system's synchronization mechanism. The following synchronization issues were taken into account:

- the number of event arguments associated to a gate and the number of partner processes involved in a synchronization,
- the combination of value offers (!) (or bangs) and variable offers (?) (or queries) in a synchronization,
- and the number of offers in an alternative choice.

To make some comparison possible between the performance of our system and that of the TOPO compiler, we used the specifications as proposed by Mañas[4]. The measurements for our system were performed on a VAX-3100 station (OS:VAX/VMS, 2.8VAPS). In each measurement the intended event was executed 1000 times. The resulting values represent the execution time per event (in ms). The plotted figures also contain the corresponding results as obtained by the TOPO compiler (directly derived from the paper[4]). The values of the TOPO compiler were measured on a Sun-4 SPARC-system 470 (OS:SunOs 4.1.1, 22RISC-MIPS). The difference in speed between both target machines was not taken into account in the following figures.

3.1 Effect of synchronization cardinality

This performance experiment addresses two issues with respect to performance; the effect of the number of event arguments associated to a gate and the effect of the number of partners involved in a synchronization.

<< Outline of the specification structure >>

```
Pr[g] || Pe[g] where
process Pr[g]: noexit :=
  Pe[g] || ... || Pe[g]     endproc
process Pe[g]: noexit :=
  g !a ... !a; Pe[g]     endproc
```

Two kinds of graphs are obtained (Fig.1). One in which the number of partners (P) varies with respect to the number of event arguments (E), and the other in which the number of event arguments varies with respect to the number of partners.

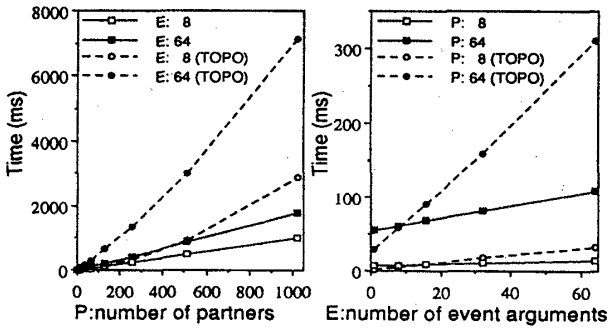

Fig. 1: Effect of the synchronization cardinality

3.2 Effect of communication pattern

This performance experiment addresses the effect of the communication pattern on the performance. That means what is the influence of the percentage of bangs with respect to queries in a synchronization.

"LOTOS 実行系の性能評価"
野村真吾†, 瀧塚孝志†, 長谷川亨†, Ron Greve‡
† 国際電信電話株式会社 研究所, ‡University of Twente

```
<< Outline of the specification structure >>
  Pbang[g] || Pquery[g] where
  process Pbang[g]: noexit :=
    Pb[g] || ... || Pb[g] endproc
  process Pquery[g]: noexit :=
    Pq[g] || ... || Pq[g] endproc
  process Pb[g]: noexit :=
    g !a ... !a; Pb[g] endproc
  process Pq[g]: noexit :=
    g ?n0:Nat ... ?ni:Nat; Pq[g] endproc
```

The results of this experiment are plotted in Fig.2 for different numbers of event arguments. The X-axis represents the percentage of queries with respect to bangs. We fixed the sum of bangs and queries to 100.

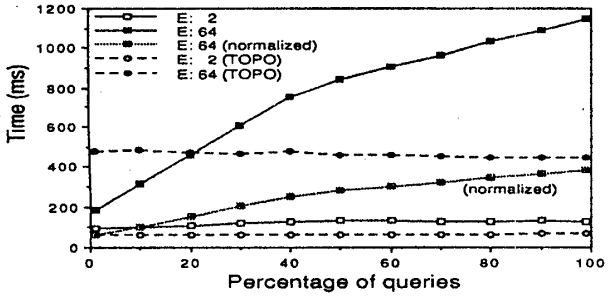

Fig. 2: Effect of the communication pattern

3.3 Effect of offer architecture

This performance experiment addresses the effect of the number of offers in an alternative choice on the performance.

```
<< Outline of the specification structure >>
  feed[g] || tree[g] where
  process feed[g]: noexit :=
    g !a;  feed[g]; endproc
  process tree[g]: noexit :=
    proposer[g] PAR_OP ... PAR_OP proposer[g]
  endproc
  process proposer[g]: noexit :=
   g !a; proposer[g] [] ... [] g !a; proposer[g]
  endproc
```

For each model of parallel operator (PAR_OP), namely interleaving (|||) and synchronization (||), results are plotted (Fig.3). In each graph the number of offers (O) varies with respect to the number of partners (P).

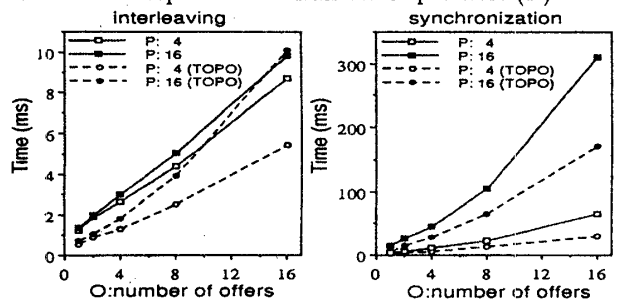

Fig. 3: Effect of the offer architecture

4. Discussions

The performance results of our LOTOS Execution System and the TOPO compiler were obtained on different target machines with a totally different architecture. If the difference of the target machines is taken into account, it turned out that our system provides a more efficient synchronization than the TOPO compiler. A significant reason for this is that our system introduced some practical devices, such as the deterministic detection of an event, which are necessary for achieving efficiency.

(1) Figure 1 shows that there is a linear cost increase with respect to both the number of partners and the number of event arguments. In case of the TOPO compiler there is a power cost increase with respect to the number of partners, and a linear cost increase with respect to the number of event arguments. The figure also shows that the increasing percentage of the graphs is lower in case of our LOTOS Execution System, especially for a large number of partners involved. For the synchronization aspect looked at in this experiment, our system thus provides efficient mechanism.

(2) There is a polynomial cost increase with respect to the percentage of queries in the communication pattern (Fig.2). This is in contrast with the TOPO compiler where the communication pattern has no noticeable impact. The reason of the polynomial cost increase is that long chains of descriptors may be built as a result of unification among queries[2]. During the unification among bangs, however, only matchings of values and their sorts are checked for efficiency reasons.

In order to make a comparison, our results are normalized in Fig.2, assuming that 1 VAPS corresponds to 2.5 RISC-MIPS. Our system shows the worst cost when the percentage of queries in a synchronization is 99%. Even then, the cost of our system does not exceed the cost of the TOPO compiler.

(3) Figure 3 shows that both compilers have a similar cost increase with respect to the number of offers. When ignoring the difference in target machines, the costs in case of our system are higher. As for our system, if a process contains an alternative choice, process switching between the scheduler and that process occurs for each event offer. The specification we used for this experiment caused many of these process switchings.

5. Conclusions

This paper discussed the performance of the LOTOS Execution system. Performance results were presented for different patterns of synchronization. It turned out that our system provides more efficient synchronization than the TOPO compiler, if the difference of the target machines is taken into account. We can expect that our system can derive an efficient implementation. The authors would like to thank Dr. K. Ono, Director, Dr. Y. Urano, Deputy Director of KDD R&D Labs, Dr. K. Suzuki, Manager of OSI Systems Group and Mr. K. Konishi, senior research engineer of KDD R&D Labs, for their helpful suggestions. It should be noted that Ron Greve joined this work under his technical training at KDD R&D Labs.

References

[1] ISO8807, "LOTOS", Feb. 1989.

[2] S. Nomura et.al.,"A LOTOS Compiler and Process Synchronization Manager", PSTV, X, IFIP, 1990.

[3] A. Mañas et.al.,"The TOPO Implementation of the LOTOS Multiway Rendezvous", Tech. rep., Dpt. Telematics, Technical Univ. Madrid, Spain, Jan. 1991.

[4] A. Mañas et.al.,"Behaviour Compiler Performance: Synthetic Specifications" , Lotosphere project ESPRIT 2304 Lo/WP2/T2.2/UPM/N0025/V01, UPM, 1991.